Software Process Assesment (SPA)

Linda H. Rosenberg, Ph.D., Unisys Government Systems, 10265 Aerospace Drive, Lanham, MD 20706,

lhr@kong.gsfc.nasa.gov

Sylvia B. Sheppard, NASA/GSFC, Code 522, Greenbelt, MD 20771

Scott A. Butler, University of Maryland, Department of Psychology, College Park, MD 20742

Abstract

NASA's environment mirrors the changes taking place in the nation at large, i.e. workers are being asked to do more work with fewer resources. For software developers at NASA's Goddard Space Flight Center (GSFC), the effects of this change are that we must continue to produce quality code that is maintainable and reusable, but we must learn to produce it more efficiently and less expensively. To accomplish this goal, the Data Systems Technology Division (DSTD) at GSFC is trying a variety of both proven and state-of-the-art techniques for software development (e.g., object-oriented design, prototyping, designing for reuse, etc.).

In order to evaluate the effectiveness of these techniques, the Software Process Assessment (SPA) program was initiated. SPA was begun under the assumption that the effects of different software development processes, techniques, and tools, on the resulting product must be evaluated in an objective manner in order to assess any benefits that may have accrued. SPA involves the collection and analysis of software product and process data. These data include metrics such as effort, code changes, size, complexity, and code readability. This paper describes the SPA data collection and analysis methodology and presents examples of benefits realized thus far by DSTD's software developers and managers.

1 Introduction

Effective management of software development projects requires continual assessment of the development process and the resulting software product. The Software Process Assessment (SPA) program of the Software and Automation Systems Branch (Code 522) of the Goddard Space Flight Center (GSFC) was established four years ago in order to promote understanding of our software development process and to assure the quality of our software products. For the purposes of this paper, terms are defined as follows: "software process" is the set of activities and methods employed in the production of software; "measurements" are raw data relating to the development effort or the software; and "metrics" are combinations of measurements used to quantify a software attribute (IEEE-Std-610.12-1990).

SPA's primary objective is to understand the effects of different life cycles, project domains, development languages, design methodologies, and management techniques on resulting software products. We are interested in developing a process model that incorporates these issues and that supports quality assurance and quality control. At present, our guide for process improvement involves tracking and analyzing daily activities in the context of our experiences and lessons learned. These analyses will benefit on-going projects by reducing development times, decreasing development costs, decreasing maintenance costs, and increasing software reliability (Baumert & McWhitney, 1992). Future development efforts will benefit by having a more accurate basis for predictions about development costs and schedules.

The fundamental premise of SPA is that metrics will not be used to evaluate programmers or project managers. To foster confidence among the programmers, each programmer and project is identified by an identification number to maintain anonymity. Through working closely with SPA personnel, project managers use the metrics to improve or evaluate current development techniques. Guidelines for using the metrics are being developed to assist managers in interpreting project results.

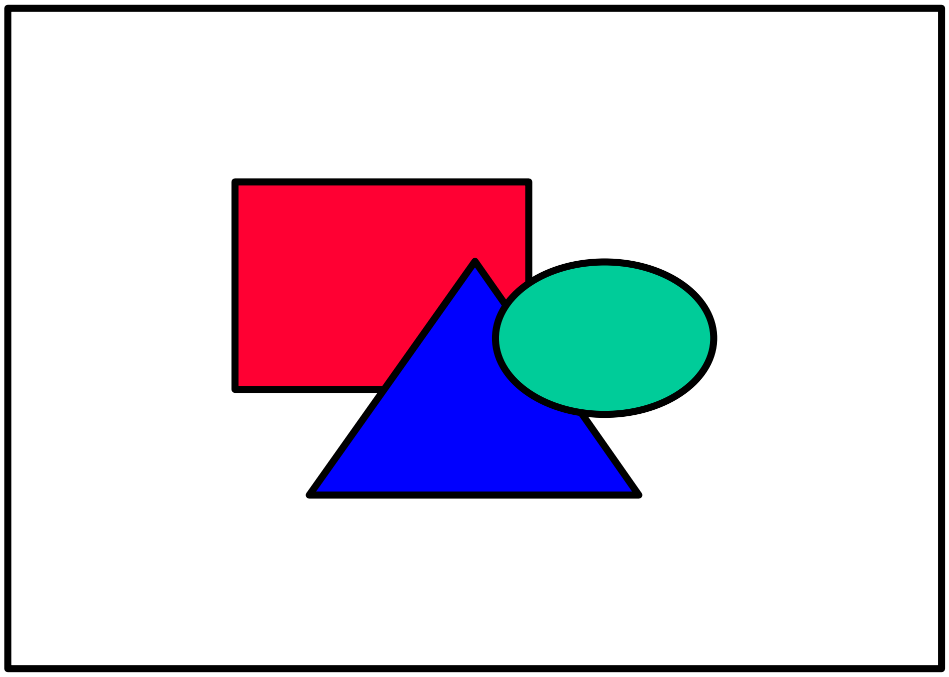

Figure 1: Measurable Components of the Software Process Assessment

2 SPA Metrics: Process, Product, and Changes

2.1 Process Metrics

SPA involves the evaluation of process and product metrics as indicated in Figure 1. To evaluate process, we focus on the application of resources, primarily personnel effort. By understanding how personnel resources are allocated in different phases, we can begin to determine how a project applied a particular life cycle model and the effects that life cycle had on the allocation of effort. This information can also assist in determining stability of requirements by tracking the amount of effort that was devoted to requirements specification. Requirement specification should occur in the initial phases of a project's life cycle; work on requirements later in the life cycle may indicate instability in the project definition (Baumert & McWhitney, 1992; Mills & Dyson, 1990).

2.2 Product Metrics

To evaluate a product, we analyze the software throughout development and after releases. Multiple analyses allow comparisons among releases and allow us to correlate effort metrics to change data. The frequency of analysis is determined by development phase and project manager requests.

Product assessments include metrics such as size, complexity, and readability (Rombach, 1990). We obtain these metrics using UX-Metric from SET Laboratories (Set Laboratories, 1990). UX-Metric produces McCabe's complexity metrics, counts GOTOs and comments, and calculates size metrics (IEEE-Std 1045-1992).

Size metrics refer to line counts, such as total lines of code (including comments and blank lines) and executable statements (measured by delimiters). Because we wish to compare metrics across different languages, we use executable statements as opposed to non-comment non-blank lines (NCNB). Executable statements are least affected by programmer style (Putnam & Myers, 1992).

Complexity metrics describe the logical structure of the individual code modules. We are initially evaluating the structure using McCabe's cyclomatic complexity (McCabe, 1976), and the extent of the use of the GOTO statement (especially in object oriented design systems) (Booch, 1991). At a later time, we will include level of nesting, fan in and fan out.

Readability metrics include the use of comments and the average length of variable names. Using comments and meaningful variable names contributes to the reader's understanding of code. Readability metrics, as well as complexity metrics, are cited in the literature as contributing to understandability of the code, an issue for code reading during development and for later maintenance of the code (Putnam & Myers, 1992).

2.3 Changes to Code

Additionally, we track the types of changes made to the code, when they were made and why they were made (Baumert & McWhitney, 1992; SEL-87-008). Errors, usability issues, and modifications to requirements are all classified as changes. In short, a change is anything that causes a modification to the code once it has been submitted to the project library. Change data are collected from the time components are entered into the project library until the completion of the development effort and, sometimes, throughout the project maintenance phase. We are also investigating correlations between the number of changes per module/file and the code metrics.

3 Data Collection

The data collection process was designed to ensure that the metrics we collected would be reliable and relevant, i.e. the data can be used to draw valid conclusions and to answer specific questions (Baumert & McWhitney, 1992). The data collection forms are non-threatening, easy-to-use, and non-intrusive. All forms are on-line and are distributed and processed electronically.

SPA uses modified versions of three forms developed at NASA Goddard's Software Engineering

Laboratory (SEL) (SEL-87-008). The forms were modified to encompass the range of activities and interests specific to the DSTD. The Personnel Resources Form (PRF) provides information about effort spent in various development activities. It is completed each week by all personnel performing either technical or management activities on a project. These activities have become an integral part of our software development process as opposed to mere adjuncts done at the discretion of the developers.

The Component Origination Form (COF) provides details about an individual software module. A COF is completed each time a component is added to the system library. One area of interest is the number of components generated "from scratch" as compared to the number that are reused (or modified and reused) from the DSTD Reuse Software Library.

The Change Report Form (CRF) describes a software change and provides a reason for the change. A CRF is completed by any person who implements a change to the system that involves modifications to components in the project-controlled source library.

4 Results

SPA data have been collected on over of thirty-five projects to date. The projects are diverse in application domain, use the waterfall or evolutionary prototyping life cycles, and are written in Ada, C, C++ or FORTRAN. Data for some projects were collected using the method described above. In other projects, completed code was obtained, but no process data were available.

4.1 Resource Analysis

The process data collection has yielded interesting results. One result is the use of metrics to drive the development of a process model. Figure 2 shows the total weekly hours by activity across the development of a C++ project currently in the third build. This chart can give management an indication of staffing requirements and can indicate the effects of events such as holidays, winter storms, and design reviews. When data from several projects of this size and type have been obtained, we hope to be able to build a model that will help estimate the staffing requirements for our specific development environment.

Besides aiding in the development of a planning profile for staffing, effort data can be used as feedback for current development efforts. One measure of the stability of a development process is the stability of activities within a phase; earlier phases should be largely completed before subsequent phases begin. For example, once a

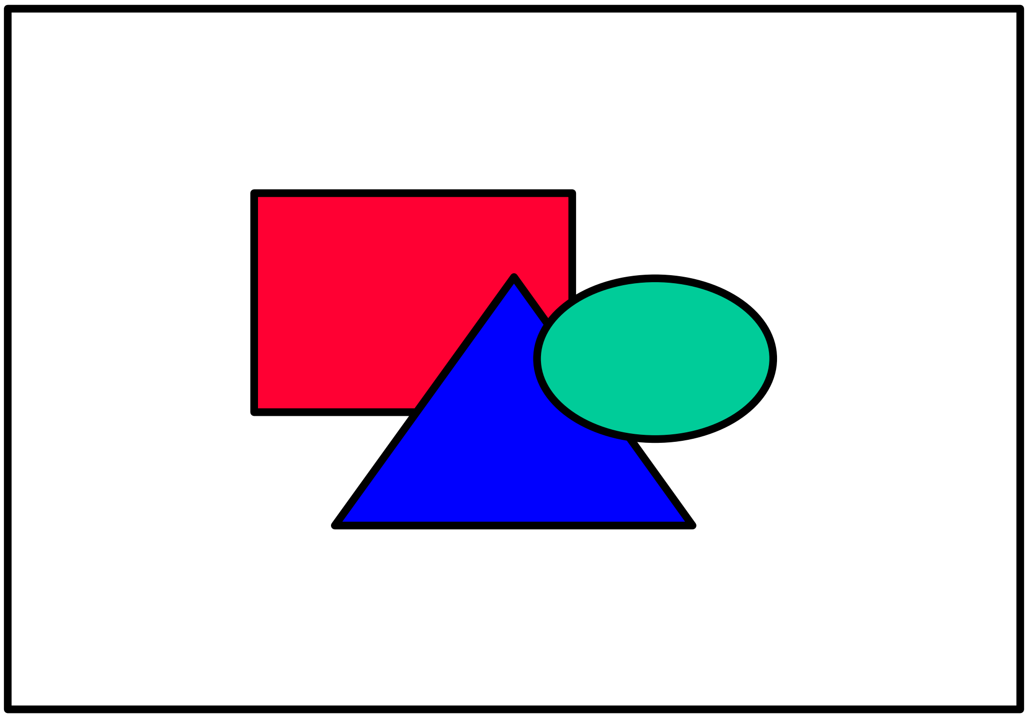

Figure 2: Hours by Week by Activity

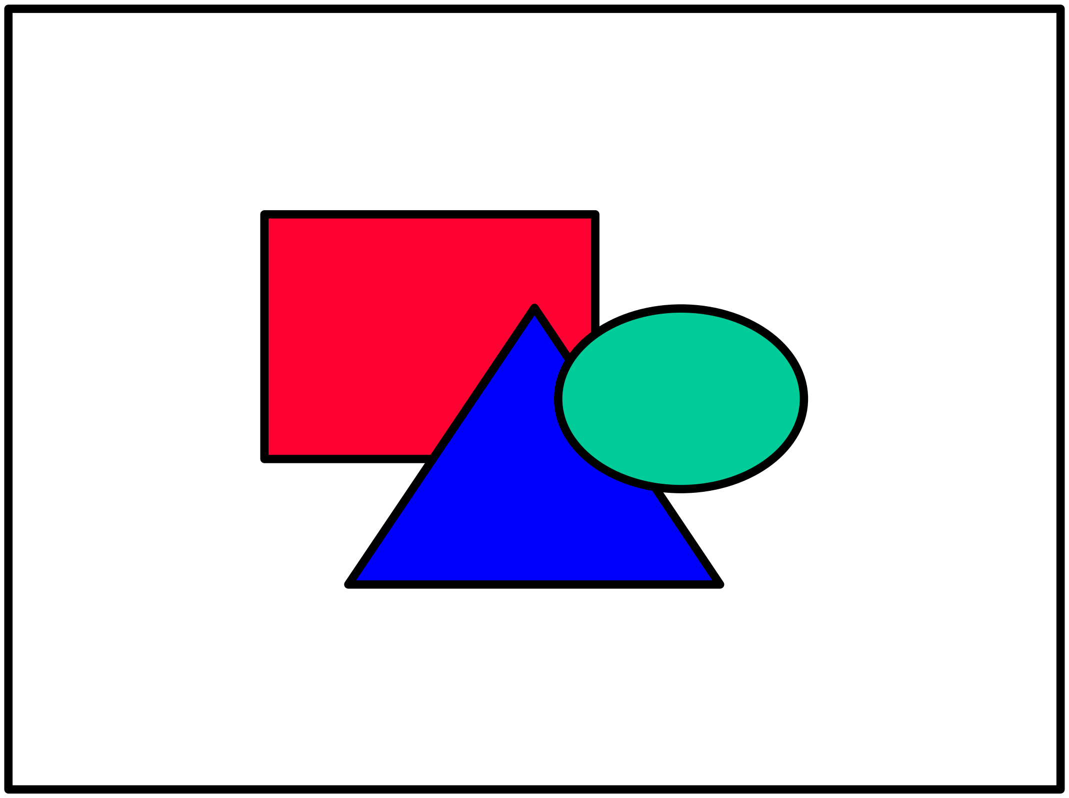

Figure 3: Total Hours per Build by Activity

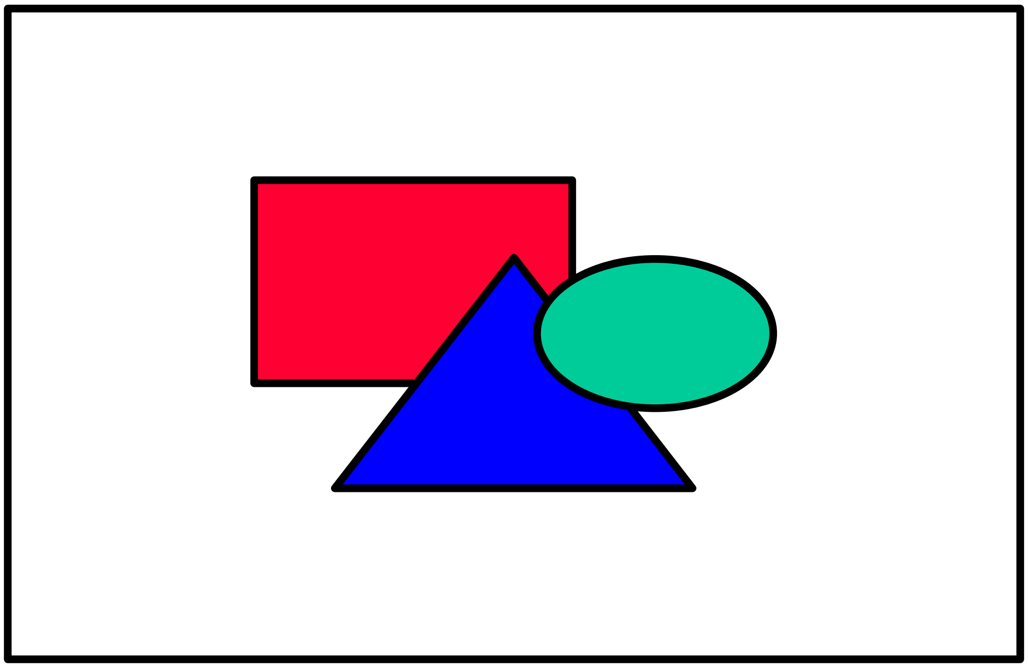

Figure 4: FORTRAN Modules

project has entered the coding phase, requirements-related activities should have been, for the most part, completed. In Figure 2, the design activity that begins on or about 9/3/93, was, in fact, in preparation for Build 2. Had this redesign been associated with Build 1, it would have been an indication of design instability and could have been costly to implement.

Figure 3 shows data from the same project, but with a more detailed breakout of life cycle activities. This graph shows that all requirement activity was completed in the first build. This is a good indicator of requirement stability. Additionally, the large design effort for Build 1 appears to have reduced the need for design in Builds 2 and 3. According to the project manager, the more difficult capabilities were added in Build 2, hence a larger amount of system testing was needed in that build.

4.2 Code Analysis

Code metrics can be used for identifying code that may be difficult to maintain and for identifying modules that may need additional testing. Modules with high complexity and/or large numbers of executable statements are prime candidates for the most extensive testing (Set Laboratories, 1990). These modules also need to be well-commented for readability (Putnam & Myers, 1992).

Figure 4 shows data for a FORTRAN project. Each square represents a module of code. This project contained 906 modules with a total of 75,537 executable statements. Most of the code was FORTRAN 77, but some was older FORTRAN IV code. This older code was difficult to maintain, but funding to rewrite it was not forthcoming. Figure 4 shows five modules (on the right-hand side of the graph) to be exceptionally high in complexity as well as being rather large, as measured by the number of executable statements. Further investigation identified those modules as part of the FORTRAN IV code. Using this chart, it was argued that code this large and complex was expensive to maintain, and an overall rewriting of the code was approved.

For projects currently under development, an effort is being made to prevent outliers such as the five that were identified in Figure 4. Analysis of modules as they are entered into the project library allows project managers to identify modules that need more testing, more extensive documentation and/or division into more manageable components.

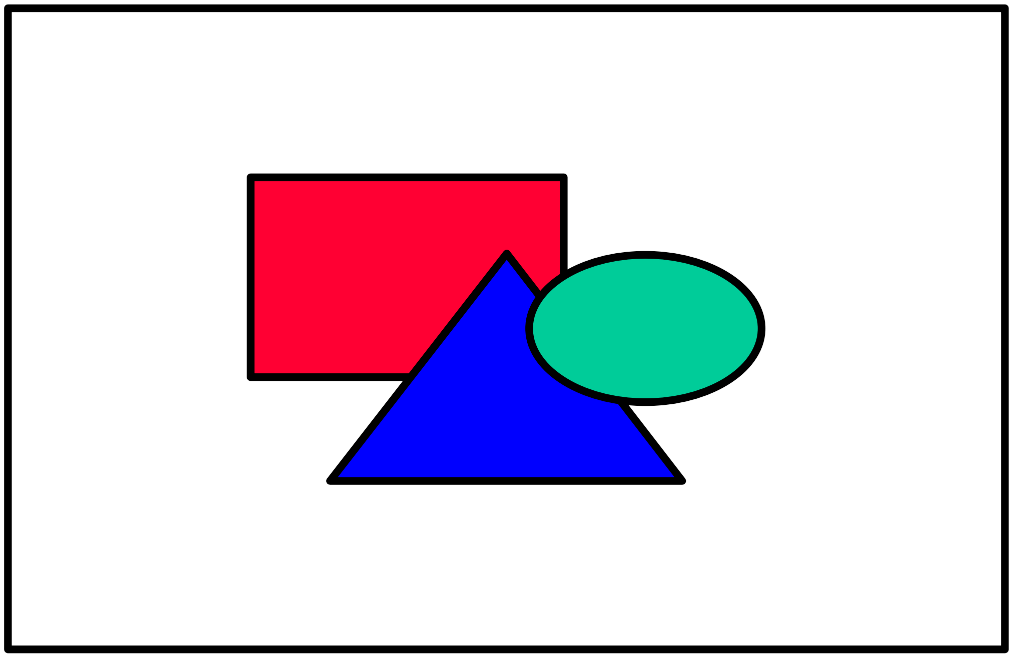

Figure 5: Number of Changes over Time by Development Phase

4.3 Software Change Analysis

Analyzing software changes can provide information about the development process as well as the product. In Figure 5, the black squares represent the cumulative changes for a C++ project currently in development. These changes may be due to planned enhancements, clarifications, requirements changes, or errors. It is expected that when this "total" curve levels off, most (if not all) errors will have been located, and the code will be ready for release. The white squares represent changes due to coding errors. In the initial phases, changes are not due to errors, but by the time of integration testing, most changes are the result of errors. Identifying trends such as this one helps us to allocate resources, both for testing and for error correction.

5 Discussion

The initial results of the SPA measurement-based process model are encouraging. We are meeting our objectives to learn about techniques in applying the life cycle in different application domains and with different languages. On the basis of management interest in the data and its application, SPA seems to be succeeding in supplying useful feedback during the development process. The metrics are also useful in identifying more efficient software development techniques.

The paragraphs that follow contain examples of how SPA feedback has helped developers address issues in the areas of design, training, budget, and quality.

Example 1:

We compared two projects done by essentially the same personnel. On the first project, personnel used diagramming for both high level design and low level design. On the second project, they used diagramming on only the high level design and instead wrote class specifications in C++, the development language. Additionally, during low-level design for the second project, they standardized on very structured development techniques involving object-oriented programming and specific call-back mechanisms. Comparing SPA data from the two projects helped to convince management that the changes in methodology had, in fact, increased productivity. The new design methodology will be continued in the future.

Example 2:

SPA metrics have been used to draw inferences about training and staffing. Information on personnel activities is being used to justify the number of hours allocated to various activities, e.g. more time spent on training or more time spent writing requirements/specifications. The analysis of an Ada project indicated that more time should have been spent training programmers to use Ada. The supposition is that if more time had been allocated early in the development cycle to learning to program in Ada, the efficiency of the project and the resulting code would have been improved.

Example 3:

Another project we studied had finished under budget and ahead of schedule. One supposition for this outcome was that a larger percentage of civil service personnel had been added than had been projected or would normally have been used on a project of this size. (Only contractors' salaries are included in the cost of a project, so in a sense civil servants are "free" labor.) By using PRF data, we were able to differentiate the number of hours and types of activities performed by contractors and civil servants. As a result, management was reassured that the early completion of the project was, in fact, due to more efficient development techniques rather than an excess of civil servants. Because of this analysis, future projects will adopt these development techniques.

Example 4:

SPA metrics have also been used to settle questions about code quality. An abbreviated development schedule caused management to question the robustness and maintainability of an application. Using the code metrics, we demonstrated that the majority of the code met the standards used at Johnson Space Center, which is known for its emphasis on software quality. Further, the metrics were used to argue successfully that portions of the code were reliable and maintainable and should not be rewritten.

6 Summary and Future Research

Metrics are often viewed by managers and programmers as threatening, but for the past four years they have been successfully collected and used to evaluate the development process model and software products in the DSTD at Goddard Space Flight Center. We attribute this success to the strict adherence to anonymity of personnel and projects and to non-intrusive data collection methods.

Although it is too early to quantify the financial benefits from these analyses, we have seen process improvements. For example, the need for training in object oriented design methods and in programming languages is determined at the start of new projects. Design and development techniques have been structured and formalized. Different testing methods are being identified and investigated.

The design and use of a measurement-driven process model has been educational. Everyone has become more aware of the structure of the development cycle and the characteristics that are related to quality program code. Through SPA, we continually evaluate our processes, making changes and improvements as necessary. Through the application of metrics, we expect the software development process to be more efficient, more predictable, and we expect higher quality products that are easier to maintain and reuse.

Our initial research used only a core metric set that focused primarily on code. There is much more to be done. We are working on correlating code metrics with discrepancy and change data in order to develop a baseline and tolerances that indicate the quality and reliability. Other code metrics, such as physical source statements, logical source statements and nesting levels are being investigated (Rombach, 1990). In the future we expect to use code metrics for certifying code before it is placed in the reuse library. We are also researching applicable metrics for other phases of the life cycle. The goal is to develop acceptability ranges for software metrics, at all phases of the life cycle, similar to those currently existing for hardware.

7 Acknowledgments

NASA's Software Engineering Laboratory (SEL) provided early and important mechanisms for understanding and managing the data collection process. Papers from the SEL, that appeared as early as 1977, have contributed greatly to the field of software engineering. We gratefully acknowledge the help of SEL personnel in getting the work cited in this paper started and in providing the data collection forms used in the first phase of these analyses.

8 References

Baumert, John H., & McWhitney, Mark S. (1992). Software measures and capability maturity model. Software Engineering Institute, Carnegie Mellon University, CMU/SEI-92-TR-25.

Booch, Grady, (1991). Object oriented design. Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA..

Software Engineering Laboratory, NASA Goddard Space Flight Center (1987). Data collection procedures for the rehosted SEL database. Software Engineering Laboratory Series, SEL-87-008.

IEEE Standard of Software Productivity Metrics, IEEE-Std 1045-1992, January, 1993.

IEEE Standard glossary of Software Engineering Terms, IEEE-Std-610.12-1990.

Mills, Harlan D., & Dyson, Peter B., (1990). Using metrics to quantify development, IEEE Software, March.

McCabe, Thomas J., (1976). A complexity measure. IEEE Transactions on Software Engineering, Vol. SE-2(4), Dec.

Putnam, Lawrence H., & Myers, Warren (1992). Measures for excellence: reliable software on time, within budget. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Rombach, H. Dieter (1990). Design measurement: some lessons learned. IEEE Software, 7(2) March.

UX-Metrics, Set Laboratories Incorporated, Oregon, 1990.